Japanese Patent Office

Patent Laying-Open Gazette

Patent Laying-Open No.	9-26876
Date of Laying-Open:	January 28, 1997
International Class(es):	G 06 F 9/30

(12 pages in all)

Title of the Invention:	Method and Apparatus for Multiplatform Stateless Instruction Set Architecture
Patent Appln. No.	7-305194
Filing Date:	October 31, 1995
Priority Claimed:	Country: U.S.A. Filing Date: October 31, 1994 Serial No. 08/332005
Inventor(s):	Paul Borrill
Applicant(s):	SUN MICROSYSTEMS, INC.

(transliterated, therefore the spelling might be incorrect)

(19)日本国特許庁（ＪＰ）　(12) **公開特許公報（Ａ）**　(11)特許出願公開番号

特開平9－26876

(43)公開日　平成９年(1997)１月28日

(51)Int.Cl.[6]	識別記号	庁内整理番号	ＦＩ		技術表示箇所
Ｇ０６Ｆ　9/30	３１０		Ｇ０６Ｆ　9/30	３１０Ｅ	

審査請求　未請求　請求項の数11　ＦＤ　（全 12 頁）

(21)出願番号　特願平7－305194

(22)出願日　平成７年(1995)10月31日

(31)優先権主張番号　０８／３３２００５

(32)優先日　1994年10月31日

(33)優先権主張国　米国（ＵＳ）

(71)出願人　591064003
サン・マイクロシステムズ・インコーポレーテッド
SUN MICROSYSTEMS, INCORPORATED
アメリカ合衆国　94043　カリフォルニア州・マウンテンビュー・ガルシア　アヴェニュウ・2550

(72)発明者　ポール・ボリル
アメリカ合衆国　95014　カリフォルニア州・カッパチーノ・マドリッド　ロード・10640

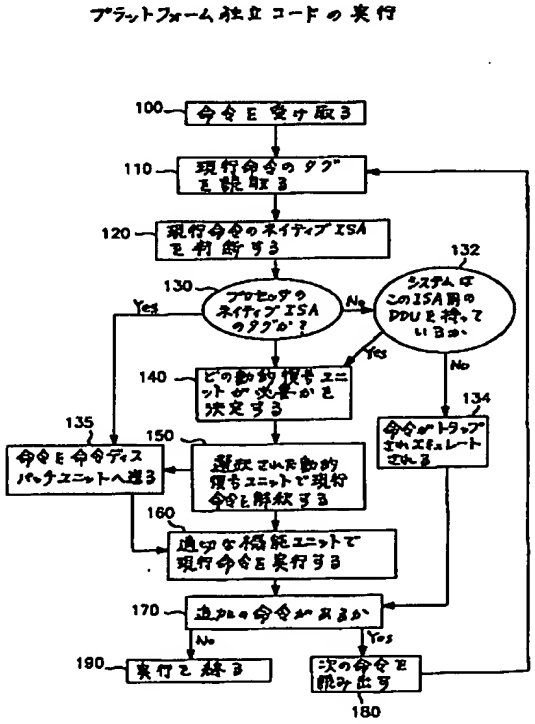
(74)代理人　弁理士　山川　政樹

(54)【発明の名称】　マルチプラットフォーム・ステートレス命令セット・アーキテクチャのための方法および装置

(57)【要約】

【課題】　複数の、通常は互換性のない命令セット・アーキテクチャ用に作成された命令を、単一の新しい統一アーキテクチャで実行することができるようにする。

【解決手段】　プロセッサは実行のための命令を受け取ると、命令を検査して、ＩＳＡタグから、命令がどの元のネイティブＩＳＡに対応しているかを判断する。対応するＩＳＡがプロセッサのネイティブＶＬＩＷ　ＩＳＡである場合、命令はプロセッサの命令ディスパッチユニットに送られ、次に実行のために少なくとも１つの機能ユニットに送られる。対応するＩＳＡがネイティブＶＬＩＷ　ＩＳＡではない場合、命令は複数の動的復号ユニット（ＤＤＵ）の１つに送られる。各ＤＤＵは命令を非ネイティブＩＳＡからネイティブＶＬＩＷ　ＩＳＡに変換する変換ルーチンによって制御される。



1

【特許請求の範囲】

【請求項１】　プロセッサとそのプロセッサに接続されたメモリとを備え、そのメモリには第１および第２の命令を操作するプログラム命令を含む複数の制御モジュールを記憶し、前記第１および第２の命令はそれぞれ第１および第２の所定の命令セット・アーキテクチャ（ＩＳＡ）に従って構成されており、第１のＩＳＡが前記プロセッサのネイティブＩＳＡで、第２のＩＳＡが前記プロセッサの非ネイティブＩＳＡであるコンピュータ・システム内で前記第１および前記第２の命令を実行するシステムにおいて、

前記プロセッサに結合され、前記第１および第２の命令を受け取る命令ルータと、

命令ルータに結合され、少なくとも前記第１の命令を受け取る命令ディスパッチユニットと、

前記命令ルータに結合され、前記第２の命令を受け取る少なくとも１つの動的復号ユニットと、

前記第１および第２の命令の各命令に結合され、それらの命令に対応するＩＳＡを識別するＩＳＡタグと、

前記第１および第２の命令セットの前記各ＩＳＡタグを読み取るタグ読取り制御モジュールと、

前記各ＩＳＡタグに対応するＩＳＡを判断するタグ識別制御モジュールと、

プロセッサのネイティブＩＳＡに対応するＩＳＡタグを有する少なくともいくつかの前記命令を前記命令ディスパッチユニットに送る第１の経路指定制御モジュールと、

所定の非ネイティグＩＳＡに対応するＩＳＡタグを有する少なくともいくつかの前記命令を前記動的復号ユニットに送る第２の経路指定制御モジュールと、

前記動的復号ユニットで受け取った命令を前記プロセッサの前記ネイティブＩＳＡに対応する修正済み命令に変換する変換制御モジュールとを含むシステム。

【請求項２】　前記修正済み命令を前記命令ディスパッチユニットに渡す第３の経路指定制御モジュールをさらに含む請求項１に記載のシステム。

【請求項３】　前記命令ディスパッチユニットに結合され、その命令ディスパッチユニットからの命令を受け取り、その命令を実行する少なくとも１つの機能ユニットと、

前記命令ディスパッチユニットで受け取った命令を前記機能ユニットに渡す第４の経路指定制御モジュールとをさらに含む請求項１に記載のシステム。

【請求項４】　前記命令ルータに結合された命令ローダと、

少なくともいくつかの前記第２の命令を前記ＩＳＡタグを含む形式に変換するローダ制御モジュールとをさらに含む請求項１に記載のシステム。

【請求項５】　プロセッサと、そのプロセッサに結合されたメモリと、前記プロセッサに結合された少なくとも

2

１つの機能ユニットと、前記プロセッサに結合された少なくとも１つの動的復号ユニットと、前記プロセッサにとって非ネイティブである第１の所定の命令セット・アーキテクチャ（ＩＳＡ）に対応する前記命令を操作する、前記メモリに記憶された前記制御プログラム・モジュールとを有するコンピュータ・システム上で命令を実行する方法において、

前記プロセッサで少なくとも１つの前記命令を受け取るステップと、

前記命令を検査して、前記プロセッサのネイティブＩＳＡに対応しているかどうかを判断し、対応している場合には前記命令を実行のために前記機能ユニットに渡すステップと、

検査した前記命令が前記第１の所定の非ネイティブＩＳＡに関係する場合には、検査した前記命令を前記ネイティブＩＳＡに対応する修正された命令に変換するステップと、

前記修正済み命令を実行のために前記機能ユニットに渡すステップとを含む方法。

【請求項６】　前記命令受取りステップの前に、少なくとも１つの前記命令を前記ネイティブＩＳＡに対応する形式に変換して前記命令を前記非ネイティブＩＳＡに対応するものとして識別するステップをさらに含む請求項５に記載の方法。

【請求項７】　命令を検査するステップの後に、前記検査済み命令が、前記プロセッサにとって非ネイティブである第２の所定のＩＳＡに対応する場合には、前記検査済み命令をエミュレートする追加のステップを含む請求項６に記載の方法。

【請求項８】　プロセッサと、そのプロセッサに結合され、命令を操作するプログラム命令を含む複数の制御モジュールを記憶するメモリとを備え、前記命令は第１および第２の所定の命令セット・アーキテクチャ（ＩＳＡ）のうちの少なくとも１つに従って構成されており、第１のＩＳＡが前記プロセッサのネイティブＩＳＡであり、第２のＩＳＡが当該プロセッサの所定の非ネイティブＩＳＡであるコンピュータ・システムで命令を実行するシステムにおいて、

プロセッサに結合され、前記命令を受け取る命令ルータと、

前記命令ルータに結合され、前記命令のうちの少なくとも第１のサブセットを受け取る命令ディスパッチユニットと、

前記命令ルータに結合され、前記命令のうちの少なくとも第２のサブセットを受け取る少なくとも１つの動的復号ユニットと、

複数の前記命令のそれぞれに対応し、複数の命令のそれぞれのネイティブＩＳＡを識別するＩＳＡ識別子と、

前記複数の命令のそれぞれが当該命令のネイティブＩＳＡであると判断するＩＳＡ識別制御モジュールと、

命令の前記第１のサブセットの少なくともいくつかを前記命令ディスパッチユニットに送り、命令の前記第２のサブセットの少なくともいくつかを前記動的復号ユニットに送る経路指定制御モジュールと、
前記動的復号ユニットで受け取った命令を前記プロセッサの前記ネイティブＩＳＡに対応する修正された命令に変換する動的復号ユニット制御モジュールとを含むシステム。
【請求項９】　前記命令ディスパッチユニットに結合された複数の機能ユニットと、
前記命令ディスパッチユニットで受け取った命令を実行のために少なくとも１つの前記機能ユニットに経路指定する命令ディスパッチユニット制御モジュールとをさらに含む、請求項８に記載のシステム。
【請求項１０】　前記プロセッサに結合され、前記プロセッサへの入力と前記プロセッサによる実行のために前記命令を受け取る命令ローダと、
複数の前記命令のそれぞれを前記プロセッサの前記ネイティブＩＳＡに対応する形式に変換する命令変換制御モジュールとをさらに含む請求項８に記載のシステム。
【請求項１１】　前記プロセッサにとって非ネイティブである第２の所定のＩＳＡをエミュレートする命令エミュレーション制御モジュールをさらに含む、請求項８に記載のシステム。
【発明の詳細な説明】
【０００１】
【発明の属する技術分野】本発明は、コンピュータ処理装置で実施され、複数の異なる命令セット・アーキテクチャ（ＩＳＡ）に準拠する命令を実行するシステムに関し、具体的には様々なＩＳＡの前記命令をステートレスで実行する処理装置に係わる。
【０００２】
【従来の技術】２進変換またはエミュレーションによって様々なタイプの命令に対応することができる複数ＩＳＡプラットフォームがいくつか提案されている。２進変換を使用すると、特定の命令セット・アーキテクチャＢ（たとえばｘ８６プロセッサ用）に準拠する命令を、まずＩＳＡ　Ａが理解する命令に２進変換することによって、第２の命令セット・アーキテクチャＡ（たとえばＳＰＡＲＣ／ＩＥＥＥ標準１７５４プロセッサ用ＩＳＡに準拠したもの）で実行することができる。次にそれらの命令を、もともとＩＳＡ　Ａ用に作成された命令であるかのように実行する。
【０００３】この手法は、ＩＳＡ　Ｂ用のプログラムの一部をＩＳＡ－Ａプロセッサで実行する場合であっても、その前にまずプログラム全体を２進変換しなければならないという点で、かなりのオーバーヘッドが伴う。これによって、ユーザが命令を処理できるまでに遅延が生ずるだけでなく、命令の処理を拘束し、２進変換が完了するまでターゲット・プロセッサが使用不能になる。さらに、所与の時間に変換中のコードの小部分のみを必要とする場合、その部分にアクセスするまでにコード全体を変換しなければならないという点で非常に非効率的である。
【０００４】さらに、２進変換は元の命令をコピーすることを必要とし、ユーザのライセンスまたはユーザが変換を行う地域の法律によってコピーが許されている場合と禁止されている場合がある。さらに、変換された命令を記憶しなければならないため、ユーザはコードの２つのコピーを保存することを余儀なくされる。変換されたコピーを廃棄する場合は、そのコードを後で使用するために、ＩＳＡ－プロセッサ上で実行するように再度変換しなければならない。
【０００５】ＩＳＡ　Ａを使用するシステムでＩＳＡ　Ｂ用に作成されたコードを実行する他の手法としてはエミュレーションがある。これは、ソフトウェア・エミュレーションとハードウェア・エミュレーションのいずれかである。ソフトウェア・エミュレーションでは、通常であればＩＳＡ　Ａ用プロセッサで認識しない命令を入力し、それを同等のＩＳＡ－Ａ命令に変換し、同等の命令をＡプロセッサで実行する必要がある。これは、Ａプロセッサのモードを ＩＳＡ　Ｂをエミュレートするように設定し、ＩＳＡ－Ｂ命令を実行し、次にＡプロセッサを再びＡモード、すなわちネイティブＩＳＡにリセットすることによって行う。
【０００６】モード変更は、プロセッサ（またはソフトウェア・エミュレーション・プログラム）に対して、次に続くデータ・ブロック（モードのリセットまで）をＩＳＡ－Ｂ命令として解釈するように命令する特別なプログラムによって行われる。たとえば、ＳＰＡＲＣプロセッサに対して、次に続くデータ・ブロックが実際にはｘ８６命令を含むことを伝える。ＳＰＡＲＣプロセッサは、各データ・ワードを１つずつ命令として解釈する。そのデータ・ブロックを命令として実行し終わると、モード設定プログラムがプロセッサのモードをそのネイティブ（たとえばＳＰＡＲＣ）モードにリセットする。
【０００７】このようにしてＩＳＡ－Ｂ命令を正常に実行することができるが、これは、特にモード変更のオーバーヘッドが重大な量となるコードの短いセクションの場合には、非効率的で時間のかかる手法であり、ＩＳＡ－Ａ命令とＩＳＡ－Ｂ命令とをコードの１つのブロック内で混在させることはできない。すなわち、異なるＩＳＡ用のコードが現れるたびに、Ａプロセッサのモードをリセットしなければならず、必然的にかなりのサイクルタイム・オーバーヘッドが生ずる。プロセッサＡでのＩＳＡ－Ｂコードの実行中に、ネイティブＩＳＡ－Ａ命令の実行を必要とする割込みを受け取った場合には特にそうである。その場合、ＩＳＡ－Ａ命令をトラップしてＩＳＡ－Ｂコードの実行を中断しなければならず、システムは正しいモードになっているかどうかを検査し、正し

5

いモードではないためにモードをモードAにリセットしなければならない。次に、トラップした命令を実行した後、システムはモードBにリセットしてＩＳＡ－Ｂコードの実行を再開する。これは法外なサイクル数を消費する。

【０００８】ハードウェア・エミュレーションは、１つまたは複数の動的復号ユニット（ネイティブ・プロセッサの内蔵部品とするか、またはアドオン・モジュールとすることができる）を使用して、ソフトウェア・エミュレーションよりも高速で実行することができる。従来のハードウェア・エミュレーションは、ソフトウェア・エミュレーションと同様に、モード設定プログラムを使用して、所与の命令セットがどのタイプのＩＳＡをエミュレートするかをプロセッサが知り、命令を適切な動的復号ユニットに送ることができるようになっている。実行速度は速くなるが、モードのリセットと、トラップと、割込みというオーバーヘッド非効率要素はソフトウェア・エミュレーションの場合と同じである。

【０００９】複数命令セット・アーキテクチャに対応するその他の手法は、増分コンパイル、すなわち１回にプログラムの一部をコンパイルする方法である。これは、プログラムの１部分だけを必要とする場合にプログラム全体をコンパイルする必要がなく、所与の設定でコンパイル時間をかなり節約することができるという利点がある。この手法は、コードを修正するたびに再コンパイルしなければならないため、自己修正コードが効率的に処理されないという明らかな欠点がある。

【００１０】同じ命令が繰り返し実行されるループでは、ソフトウェア・エミュレーションまたはハードウェア・エミュレーションを行うと、エミュレーション・プロシージャはコマンドが現れるたびに各コマンドを新たに解釈するため、同じコード・ブロックが何度も再コンパイルされることになる。増分コンパイルは、コードの自己修正が起こったとき、同じコンパイルが繰り返される可能性がある。このコンパイルの反復は相当なプロセッサ・サイクルの無駄である。

【００１１】

【発明が解決しようとする課題】ＳＰＡＲＣ、ｘ８６、ＰｏｗｅｒＰＣの各ＩＳＡおよびそれぞれのオペレーティング・システム（Ｓｏｌａｒｉｓ、ＤＯＳ、ＭａｃＯＳ）など、市場における複数命令セット・アーキテクチャの絶え間のない開発により、ユーザがこれらの様々な命令セット・アーキテクチャ用に開発されたアプリケーションを単一のハードウェア・プラットフォーム上で、不要な命令ブロックのコンパイルや、命令の非効率的な複数回コンパイル、モード設定に伴うオーバーヘッドがない方式で実行できることがますます重要になっている。複数命令セット・アーキテクチャに効率的に対応できるだけでなく、異なる命令セットの命令が入ったプロシージャを混在させることができることによって、様々

6

なＩＳＡ用に作成された多くのプログラムのうち最良のものを利用することができるシステムがあれば、特に有用であろう。

【００１２】

【課題を解決するための手段】本発明は、複数の、通常は互換性のない命令セット・アーキテクチャ用に作成された命令を、単一の新しい統一アーキテクチャで実行することができるようにする。所与の３２ビット・アーキテクチャ用の命令を、「外来」コードのネイティブＩＳＡを示すＩＳＡタグを構成する所定のビット・ブロックが含まれたもう１つの３２ビット・ワード、すなわちコードが書き込まれた３２ビット命令セット・アーキテクチャと結合する。この６４ビット命令ストリーム全体を「ホーム」プロセッサが受け取り、そのＩＳＡタグに従って各命令を実行する。

【００１３】したがって、ホーム・プロセッサは６４ビット命令セット・アーキテクチャを使用し、前述のようにして任意の３２ビット・アーキテクチャの命令を、再コンパイルやソフトウェア・エミュレーションなしでリアルタイムで実行することができる。これによって、前述の方法に関するオーバーヘッドが減少するほか、プログラマーは、インポートされる各コード・ブロックに適切なタグを付けるだけで、１つのアプリケーションで複数のＩＳＡのサブルーチン、ソフトウェア・モジュール、およびオブジェクトを自由に利用することができる。本発明は、ＶＬＩＷ（超長命令語）アーキテクチャに特に適しており、このアーキテクチャによって真のステートレス複数ＩＳＡシステムが実現される。

【００１４】

【発明の実施の形態】本発明のシステムは、プロセッサ３０とメモリ４０を有する中央演算処理装置（ＣＰＵ）２０を備えた図１に示すシステム１０のような既存のコンピュータ・システムで実施するように設計されている。通常、たとえばディスク・ドライブやテープドライブ、ＣＤ－ＲＯＭなど、記憶媒体５０のような１つまたは複数の記憶媒体を使用する。ＣＰＵ２０には命令ローダ６５を介して入出力装置６０が接続されており、これにはキーボード、モニタ、マウスなどの入出力周辺装置が含まれる。したがって本発明のシステムは、後述の例外を除き、従来のハードウェアを使用して実施することができる。

【００１５】本明細書では、本発明について、１つのプラットフォームで複数の異なる命令セット・アーキテクチャ用に作成された命令をステートレスに実行する実施形態、具体的には、６４ビット・プラットフォームで３２ビット命令を実行する実施形態で説明する。元の非ネイティブ命令セット・アーキテクチャよりも長い命令を使用して、１つのプラットフォーム上でのどのような命令セットの実行にも等しく適用可能である。これは、元のサイズが異なる様々なセットであっても可能である。

本発明は、メモリ４０に記憶されているオペレーティング・システム、すなわち、後述する方法の各ステップを実行するためにメモリに記憶された制御命令、一連の命令、あるいは１つまたは複数の制御命令を含むプログラム・モジュールの形態で実施するのが最もよい。本明細書では、「制御モジュール」とは、本発明の１つまたは複数のステップを実行するために構成された、任意のプログラム、命令のセット、プログラム・ルーチンなどを指す（図７を参照）。

【００１６】図２に、それぞれが１つまたは複数の制御命令を含んだ制御命令、一連の制御命令、またはプログラム・モジュール７０ないし７８の、これらは３２ビットの命令である、プロセッサ８０への流れを図示する。前述のように、プロセッサ８０のネイティブＩＳＡ用に作成された命令でない場合、これらの命令を実行するにはいくつかの異なる方法がある。命令７０ないし７８の上下の点は、所与のプログラム、ルーチン、または同様のものの中には一般にさらに多くの命令があることを示している。

【００１７】図３に、本発明のプロセッサ９０に送られる、命令７０ないし７８が変形された命令７０Ａないし７８Ａの流れを図示する。各命令には、図３ではビット３２ないし６３として図示されている追加の３２ビットが付加されている。実施形態に示すように、上位３ビット（またはその他の所望のビット数）は、各命令のネイティブＩＳＡを示すタグを構成する。３ビットによって、８つの異なる命令セット・アーキテクチャが識別されるため、タグＴ０ないしＴ８（それぞれ命令７０Ａないし７８Ａに対応する）のビット数は、対応する命令セット・アーキテクチャの数によって異なる。

【００１８】図４（図５も参照）のボックス１００で、プロセッサ９０が命令を受け取ると、命令は命令ルータ２０５に受け入れられ、それがネイティブＩＳＡ（この例ではＶＬＩＷなど）の命令である場合、命令ルータは命令ディスパッチユニット２００に直接送られる。受け取った命令が、ＤＤＵを使用することができる所定のいくつかのＩＳＡのうちの１つに関係する命令である場合は、命令ルータ２０５によって適切なＤＤＵに送られ、命令ディスパッチユニット２００が認識することができる命令に変換される。いずれの場合も、命令は次に、実行のために適切な機能ユニット（２８０など）に送られる。機能ユニット（ここでは例示のためにそのうちの３つが示されているが、任意の数とすることができる）としては、浮動小数点ユニット、フェッチユニット、分岐ユニット、算術演算ユニットなどがある。

【００１９】受け取った命令の適切な経路指定は次のようにして行われる。ルータ２０５で命令が受け取られると、それに対応するタグが読み取られ、その命令のネイティブＩＳＡが判断される（ボックス１１０および１２０参照）。タグは事前割り当てされた意味を有する。たとえば、０００はプロセッサ９０による変換が不要なネイティブ・プロセッサのＩＳＡを指すものとすることができる。表１にコード体系例を示す。

表１

コード	ネイティブ・プロセッサ
０００	ＶＬＩＷ
００１	ＳＰＡＲＣ
０１０	ＰｏｗｅｒＰＣ
０１１	ｘ８６
１００	（プロセッサ５）
１０１	（プロセッサ６）
１１０	（プロセッサ７）
１１１	（プロセッサ８）

【００２０】プロセッサ９０が命令７０Ａを受け取ると（図３および図５参照）、プロセッサは、その命令が、この例ではネイティブＶＬＩＷ命令、すなわちシステムのＶＬＩＷプロセッサ用とコードされた命令であると判断する。図４のボックス１３０を参照されたい。命令７０Ａは、図４のボックス１３５で示すようにルータ２０５（図５参照）によって命令ディスパッチユニット２００に送られ、実行のために適切な機能ユニット２８０、２９０、または３００に送られる（ボックス１６０）。

【００２１】次にシステムは、追加の命令があるかどうかを判断し（ボックス１７０）、ある場合には、次の命令が読み取られ（ボックス１８０）、ステップ１１０に戻って次の命令のタグが読み取られる。

【００２２】次の命令は、この時点では命令７１Ａであり、ネイティブＶＬＩＷプロセッサ用にコードされているものと判断され、したがって命令７０Ａと同様にして実行される。これは命令７２Ａについても行われ、次に命令７３Ａが現れると、そのタグがステップ１１０で前述のように読み取られる。

【００２３】ステップ１２０で、命令７３Ａのネイティブ ＩＳＡがサン・マイクロシステムズ社のＳＰＡＲＣ ＩＳＡであることが判断される。したがってこ今度はステップ１３２に進み、識別されたＩＳＡのＤＤＵがシステムに組み込まれているかどうかが判断される。答えが肯定の場合ステップ１４０に進む。否定の場合はステップ１３４に進み、トラップされてソフトウェア・エミュレーション・ステップでエミュレートされる。

【００２４】この例で、システムに適切なＤＤＵが存在するものとすると、命令はルータ２０５によってその適切な動的復号ユニット２１０ないし２７０に送られる。３ビットのタグによって対応することができる１個のネイティブ・プロセッサと７個の「非ネイティブ」プロセッサの場合、７個の動的復号ユニット（ＤＤＵ）が存在することが可能になる。この実施形態の動的復号ユニットの数が７個より少ない場合、残りのＩＳＡはトラップしてソフトウェアでエミュレートすることができ、したがってハードウェア実行速度とオンチップ資源の経済性

との間で効果的なトレードオフが可能になる。この特徴については以下で詳しく説明する。

【００２５】ＤＤＵの機能は、受け取った命令を、ネイティブ・プロセッサ９０が認識できる命令に変換することである。これには、命令を非ネイティブＩＳＡからネイティブＩＳＡに１対１で変換し（これは参照テーブルによって行うことができる）、複合命令をいくつかのネイティブＩＳＡ命令に変換し（たとえば複合ｘ８６命令はいくつかのＶＬＩＷ命令に変換することができる）、非ネイティブＩＳＡの条件コードをネイティブＩＳＡの条件コードとして実行することなどが必要である。様々なＩＳＡ用のアプリケーションを開発する当業者なら、このようなＤＤＵを簡単に実施することができよう。

【００２６】本発明で実行される機能およびステップは、前述のように、メモリ４０（図７参照）に記憶された個別の対話型命令制御モジュールとして実施されるのが好ましい。図７に示すモジュールの機能は以下の説明から明らかになろう。数個のモジュールを結合して１つのモジュールとすることができ、本発明の個別に識別可能な任意の機能のために必要であればその他のモジュールも組み込むことができる。

【００２７】したがって、図４のボックス１４０で、命令７３ＡをＤＤＵ１などの適切なＤＤＵ（図５の２１０）に送るべきかどうかが判断される。図４のボックス１５０で、ＤＤＵ１への命令の送信と、ＤＤＵ１での解釈が示されている。ＤＤＵ１は、命令を１つまたは複数のＶＬＩＷ命令に変換し、図４のボックス１３５に示されているように命令ディスパッチユニット２００（図５）に渡す。ディスパッチユニット２００は必要に応じて、命令を実行のために適切な機能ユニットに渡す（ボックス１６０）。

【００２８】ボックス１７０で、命令ストリーム内に追加の命令があることがわかり、ステップ１８０に進んで命令７４Ａが読み取られる。ボックス１１０でそのタグが読み取られ、この場合もＳＰＡＲＣ命令であると判断される。すなわち、そのタグ「００１」（図３）から、ＤＤＵ１に送るべきであると判断される。

【００２９】次の２つの命令７５Ａおよび７６ＡはネイティブＶＬＩＷ命令（コード「０００」を有する）であり、したがって命令ルータ２０５は命令ディスパッチユニット２００に渡す。命令７７Ａは、「１１０」というタグによってｘ８６プロセッサ（前記の表１参照）に属するものであることがわかり、ルータ２０５によって適切なＤＤＵ、たとえばＤＤＵ３（図５の２３０）に送られる。

【００３０】命令７８ＡもネイティブＶＬＩＷコマンドであることがわかり、ディスパッチユニット２００（図５参照）に直接送られる。それ以降の命令もそれぞれ同様にして扱われる。

【００３１】このシステムによって、任意の多数の命令セット・アーキテクチャ用に作成されたコードの命令を動的に（リアルタイムで）実行する真にステートレスな方式が実現されることがわかる。これによって、プログラマーは１つのアプリケーションで様々なＩＳＡのルーチン、モジュール、およびオブジェクトを使用して、きわめて柔軟性のある方式で、所望の場合は行単位で、アプリケーションを作成することができる。そのように作成した場合、実行時に処理上の不利はなく、事前コンパイルまたはモード設定に伴うオーバーヘッドもない。

【００３２】図５の命令ルータ２０５、命令ディスパッチユニット２００、ＤＤＵ２１０ないし２７０および機能ユニット２８０ないし３００は、通常はプロセッサ９０の内蔵部品となる。しかし、ユーザの選択システム・アーキテクチャによっては、これらのいずれも別個の要素として設計することができる。本明細書では、これらがプロセッサの内蔵部品であるか、特定の場合には主プロセッサとは別個のモジュールであるかを問わず、これらを一般にプロセッサに「結合された」ハードウェア・モジュールと呼ぶ。後述するように、ルータ２０５およびユニット２００ないし３００はすべて、メモリに記憶されている命令または命令モジュールによって制御される。これらの命令または命令モジュールは、着信命令を必要に応じて受信、検査、経路指定、変換およびその他の方法で操作する。

【００３３】図５に示すＤＤＵは、図のようにプロセッサに組み込むか、または後で付加できるプラグイン・モジュールとすることができる。したがって、本発明のシステムは、まず最初に所与の数のＩＳＡに対応するように構成することができ、モジュールを追加してネイティブ・システムのＩＳＡをそれに応じて再構成することによって、その後の命令セット・アーキテクチャにも対応することができる。

【００３４】上記で示唆したように、本発明のシステムは従来のシステムと組み合わせて最大限の柔軟性を得ることができる。たとえば、使用可能なＤＤＵがない所与のＩＳＡ用のコードを実行したい場合は、そのコードのために図４のボックス１３２ないし１３４で示してあるように、従来のソフトウェア・エミュレーションを使用することができる。これらのボックスでは、システムがＤＤＵを使用することができない命令のために従来のソフトウェア・エミュレーションが実行される。したがって、入力コードにはソフトウェア・エミュレートを行う所与のＩＳＡのための命令ブロックを組み込むことができると同時に、残りの命令には前述のようにネイティブＩＳＡを示すためのタグを付けることができる。同様に、２進変換またはハードウェア・エミュレーションを、本発明を利用するように構成された命令とともに使用することもできる。ただし、所与のＩＳＡのためにＤＤＵを使用することができれば、前述のように本発明の実施形態でそれを使用することは簡単なことであるた

め、後者の方法が用いられることは少ないと思われる。

【００３５】図６に、所与のＩＳＡ用の命令ブロックを、本発明のシステムで実行するのに適した図３に図示する構造に従った命令に変換する方法を示す。この方法は、コンピュータ・システムのメモリに記憶された命令によって行われ、図１に示すローダ６５を使用して実行される。

【００３６】ステップ３１０（図６参照）で非ネイティブ命令が受け取られてローダ６５に入れられる。ローダにはそれらの命令がどのＩＳＡ用に作成されたかに関する情報が提供される。各命令には、図３に図示する命令ストリームの左側に示されている３２ビット構造体（ビット３２ないし６３）が付加され（ステップ３２０）、それぞれの場合について非ネイティブＩＳＡの適切なタグが入れられる（ステップ３３０）。命令はこれによって修正され、６４ビット命令に変換されて、ステップ３４０で修正済みの形式で記憶される。これで命令は前述のようにして実行することができる。

【００３７】このような非ネイティブ・コードのブロックが変換され、実行されるとき、本発明のシステムによって適切なアドレス指定が自動的に行われる。図２で、命令７０はアドレス「ｎ」、命令７１はアドレスｎ＋４（８バイト・ワードとした場合）のようになっているとする。変換後、命令は図３に示す構造になり、変換された命令７０Ａは（ＶＬＩＷ）アドレス「ｎ」を有するが、命令７１Ａはｎ＋４ではなくアドレスｎ＋８を有し、命令７２Ａはｎ＋８ではなくアドレスｎ＋１６を有するというようになる。これは、特に相対アドレスを使用する場合には問題になることがあるため、本発明のシステムは、３２ビット命令セットからの命令の変換時に、６４ビット命令セットによって占有される追加アドレス空間を隠すユニットを備える。これは、各アドレスが１バイトではなく２バイトを識別することができるようにすることによって行うことができる。これは、ネイティブＩＳＡには見えることになるが、外来ＩＳＡには、各アドレスによって識別される記憶バイトがそれ以上あることはわからないため、６４ビット命令フィールドの上位３２ビットは外来ＩＳＡから隠される。

【００３８】元の非ネイティブ・コードは、図６に関して述べたようにして修正されると、正常に実行するためのそれ以上の修正は必要としない。所与のアドレス指定体系にとって、本発明のシステムのためにアドレスを適切に変換するのに必要なコードを生成することは簡単なことである。たとえば、この例では、値（ｎ＋Ｚ）を持っている前のアドレスが、値（ｎ＋２*Ｚ）を持つアドレスに変換される。ここでＺは０、４、８、１２などのオフセットであり、すなわち、オフセットは０、８、１６、２４のように２倍になる。

【００３９】まずリアルタイムでＤＤＵによって変換することにより、すべての命令がほとんどネイティブ命令と同様に実行されるため、非ネイティブ・プラットフォーム上で自己修正命令を実行するという前述の問題は、本発明によって解決されることが認められよう。したがって、コード修正は、２進変換またはエミュレーションで生ずる再コンパイルやその他のオーバーヘッドなしで、すべて自動的に行われる。

【００４０】これらの６４ビット命令が占める追加空間によって、命令キャッシュ内の占有メモリ帯域幅および空間の量が増えることになる。しかし、ＶＬＩＷプロセッサはいずれにしてもより大きなレジスタ・ファイルへのアクセスおよび命令符号化を可能にするために、３２ビットよりもはるかに大きな命令を必要とするため、これが問題になることはないと思われ、しかも命令が示す一時的、空間的局所性により、命令参照の大半がオンチップ・キャッシュにとどまり、メモリへの追加帯域幅が使い果たされることはない。したがって、本発明の命令の大きさの増大によるキャッシュ・ミスの増加は認めうるほどはないと同時に、直接マルチＩＳＡシステムによってかなりのサイクルが節約される。

【図面の簡単な説明】

【図１】　本発明を実施することができるコンピュータ・システムのブロック図である。

【図２】　プロセッサへの３２ビット命令の流れを示す図である。

【図３】　プロセッサへの本発明の６４ビット命令の流れを示す図である。

【図４】　本発明の好ましい方法を示すフロー・チャートである。

【図５】　本発明を実施するための、プロセッサ内の動的復号ユニットおよび機能ユニットを図示したブロック図である。

【図６】　従来の命令のブロックを、本発明に従って実行するための命令に変換する方法を示すフロー・チャートである。

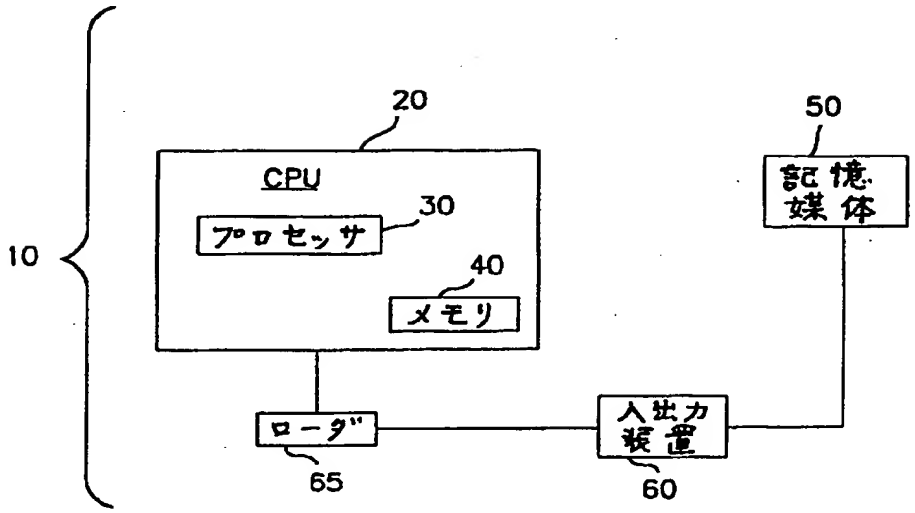
【図７】　本発明の好ましい実施形態の命令制御モジュールを図示したブロック図である。

【符号の説明】

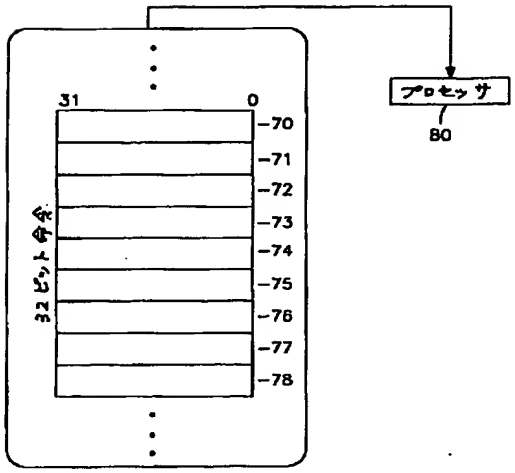
１０　システム
２０　中央演算処理装置
３０　プロセッサ
４０　メモリ
５０　記憶媒体
６０　入出力装置
６５　命令ローダ
７０　制御命令
７０Ａ　命令
８０　プロセッサ
９０　プロセッサ
２００　命令ディスパッチユニット
２１０　動的復号装置

２０５　命令ルータ　　＊　＊２８０　機能ユニット

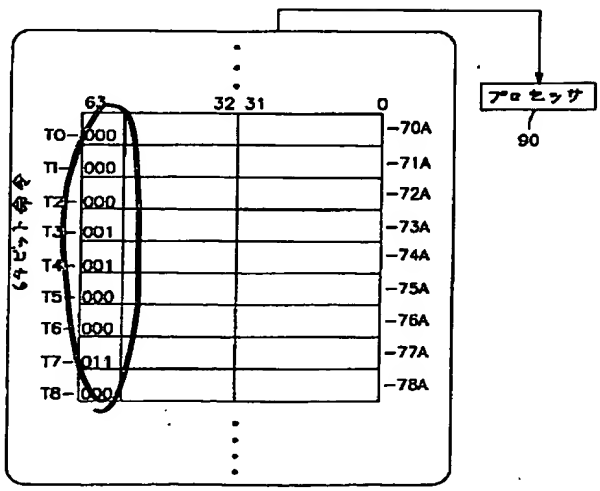
【図１】



【図２】

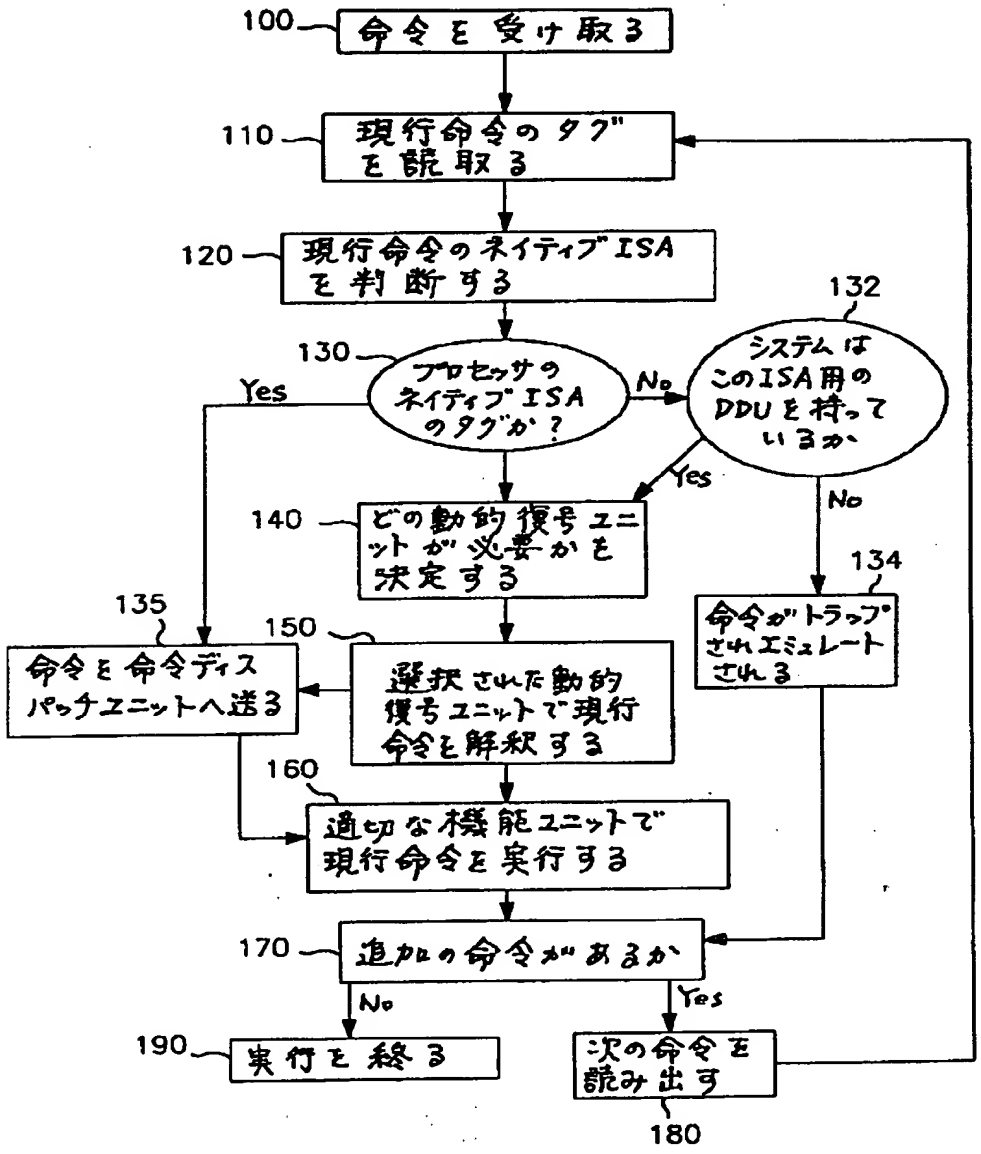


【図３】

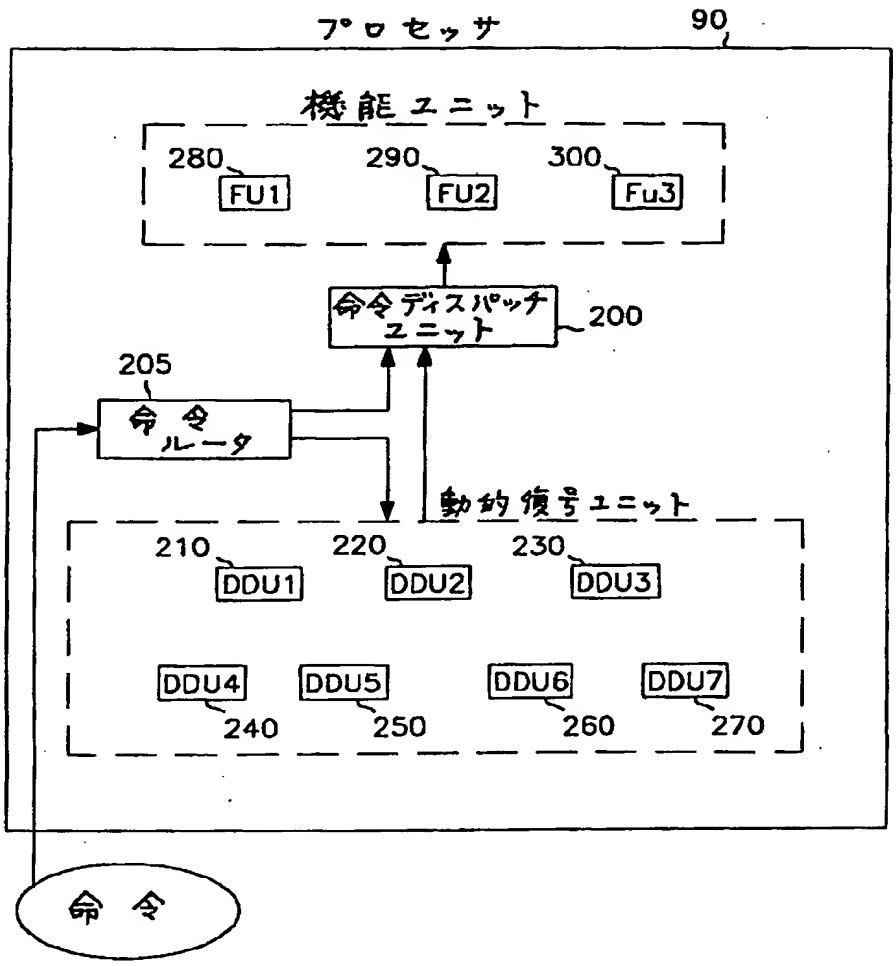


【図４】

プラットフォーム独立コードの実行

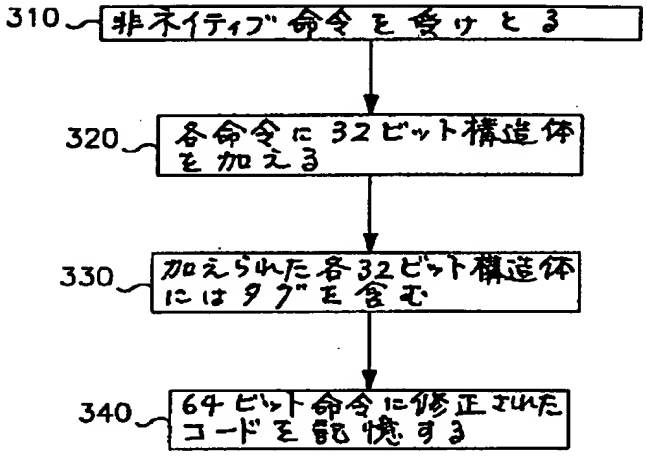


【図5】

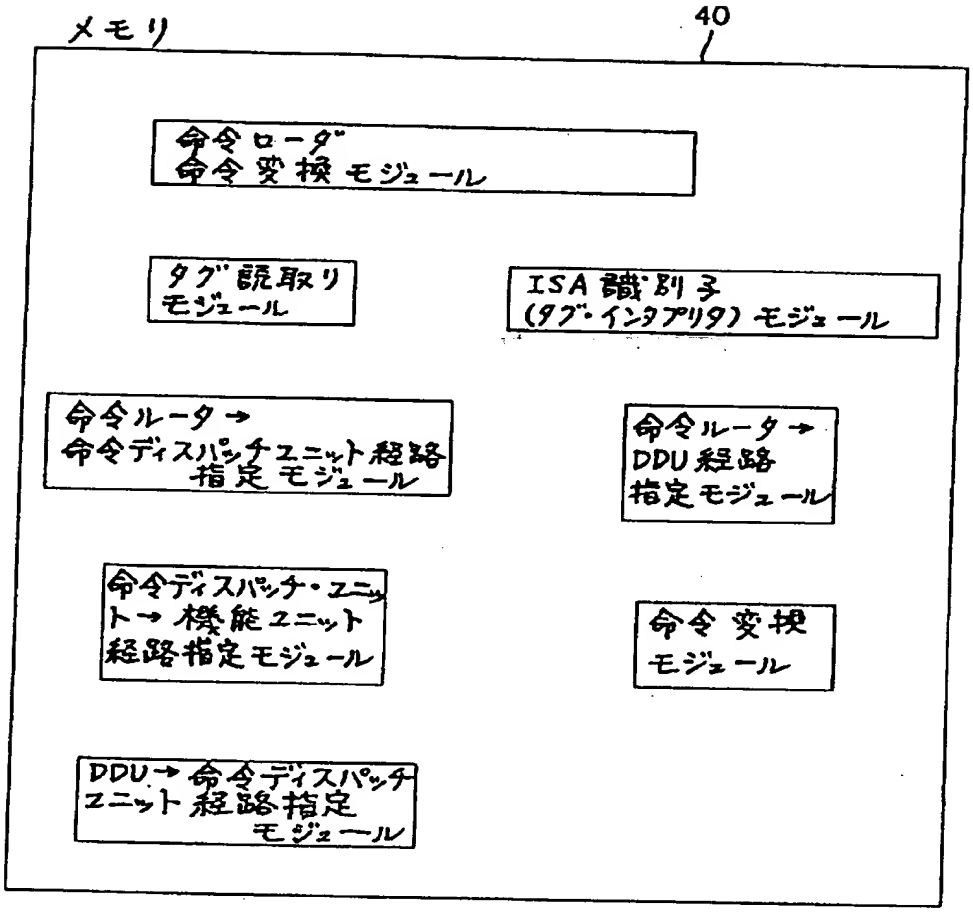

プロセッサ
90
機能ユニット
280
FU1
290
FU2
300
Fu3
命令ディスパッチユニット
200
205
命令ルータ
動的復号ユニット
210
DDU1
220
DDU2
230
DDU3
DDU4
240
DDU5
250
DDU6
260
DDU7
270
命令

【図6】

OS-Aコードからプラットフォーム独立コードへの変換



【図7】


メモリ
40
命令ローダ
命令変換モジュール
タグ読取り
モジュール
ISA識別子
(タグ・インタプリタ)モジュール
命令ルータ→
命令ディスパッチユニット経路
指定モジュール
命令ルータ→
DDU経路
指定モジュール
命令ディスパッチ・ユニ
ット→機能ユニット
経路指定モジュール
命令変換
モジュール
DDU→命令ディスパッチ
ユニット経路指定
モジュール